---

title: SUAVE Standard Library slug: /tools/suave-std description: Contracts and libraries to supercharge SUAPP development. keywords: - application - build - suave - solidity

---

# Suave-std

Suave Standard library (SUAVE-STD) is a collection of helpful contracts and libraries to build Suapps.

## Installation

To install with Foundry:

bash forge install flashbots/suave-std

## Libraries

### Transactions.sol

Helper library that defines types and utilities to interact with Ethereum transaction types.

#### Example usage

Encode an EIP155 transaction:

```solidity import "suave-std/Transactions.sol";

contract Example { function example() public { Transactions.EIP155 memory legacyTxn0; // fill the transaction fields legacyTxn0.to = address(0x095E7BAea6a6c7c4c2DfeB977eFac326aF552d87); legacyTxn0.gas = 50000; // ...

    // Encode to RLP
    bytes memory rlp = Transactions.encodeRLP(legacyTxn0);

    // Decode from RLP
    Transactions.EIP155 memory txn = Transactions.decodeRLP_EIP155(rlp);
}

} ```

Sign an EIP-1559 transaction:

```solidity import "suave-std/Transactions.sol";

contract Example { function example() public { string memory signingKey = "b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f291";

    Transactions.EIP1559Request memory txnRequest;
    txnRequest.to = address(0x095E7BAea6a6c7c4c2DfeB977eFac326aF552d87);
    txnRequest.gas = 50000;
    txnRequest.maxPriorityFeePerGas = 10;
    // ...

    Transactions.EIP1559 memory signedTxn = Transactions.signTxn(txnRequest, signingKey);
}

} ```

### Context.sol

Helper library to interact with the Suave context in the MEVM.

Available functions:

- confidentialInputs(): Returns the confidential inputs of the offchain request.
- kettleAddress(): Address of the kettle that is executing the offchain request.

#### Example usage

```solidity import "suave-std/Context.sol";
```

contract Example { function example() public { bytes memory inputs = Context.confidentialInputs(); address kettle = Context.kettleAddress(); } } ```

## Gateway

Helper library to interact with contracts from other chains.

### Example usage

```solidity import "suave-std/Gateway.sol";

contract Example { function example() public { // query the beacon chain deposit contract Gateway gateway = new Gateway("http://", address(0x00000000219ab540356cBB839Cbe05303d7705Fa)); DepositContract depositContract = DepositContract(address(gateway));

    bytes memory count = depositContract.get_deposit_count();
}

}

interface DepositContract { function get_deposit_count() external view returns (bytes memory); } ```

## protocols/MevShare.sol

Helper library to send bundle requests with the Mev-Share protocol.

### Example usage

```solidity import "suave-std/protocols/MevShare.sol"; import "suave-std/Transactions.sol";

contract Example { function example() public { Transactions.EIP155 memory legacyTxn0; // fill the transaction fields legacyTxn0.to = address(0x095E7BAea6a6c7c4c2DfeB977eFac326aF552d87); legacyTxn0.gas = 50000; // ...

    bytes memory rlp = Transactions.encodeRLP(legacyTxn0);

    MevShare.Bundle memory bundle;
    bundle.bodies = new bytes[](1);
    bundle.bodies[0] = rlp;
    // ...

    MevShare.sendBundle("http://<relayer-url>", bundle);
}

} ```

## protocols/EthJsonRPC.sol

Helper library to interact with the Ethereum JsonRPC protocol.

### Example usage

```solidity import "suave-std/protocols/EthJsonRPC.sol";

contract Example { function example() public { EthJsonRPC jsonrpc = new EthJsonRPC("http://..."); jsonrpc.nonce(address(this)); } } ```

## protocols/ChatGPT.sol

Helper library to send completion requests to ChatGPT.

```solidity import "suave-std/protocols/ChatGPT.sol";

contract Example { function example() public { ChatGPT chatgpt = new ChatGPT("apikey");

    ChatGPT.Message[] memory messages = new ChatGPT.Message[](1);
    messages[0] = ChatGPT.Message(ChatGPT.Role.User, "How do I write a Suapp with suave-std?");

    chatgpt.complete(messages);
}

} ```

# Forge integration

In order to use forge, you need to have a running Suave node and the suave binary in your path.

To run Suave in development mode, use the following command:

bash $ suave --suave.dev --suave.eth.external-whitelist='*'

Then, your forge scripts/test must import the SuaveEnabled contract from the suave-std/Test.sol file.

```solidity import "forge-std/Test.sol"; import "suave-std/Test.sol"; import "suave-std/suavelib/Suave.sol";

contract TestForge is Test, SuaveEnabled { address[] public addressList = [0xC8df3686b4Afb2BB53e60EAe97EF043FE03Fb829];

function testConfidentialStore() public {
    Suave.DataRecord memory record = Suave.newDataRecord(0, addressList, addressList, "namespace");

    bytes memory value = abi.encode("suave works with forge!");
    Suave.confidentialStore(record.id, "key1", value);

    bytes memory found = Suave.confidentialRetrieve(record.id, "key1");
    assertEq(keccak256(found), keccak256(value));
}

} ```
```

## Confidential inputs

Use the setConfidentialInputs function to set the confidential inputs during tests.

```solidity import "forge-std/Test.sol"; import "src/Test.sol"; import "src/suavelib/Suave.sol";

contract TestForge is Test, SuaveEnabled { function testConfidentialInputs() public { bytes memory input = hex"abcd"; ctx.setConfidentialInputs(input);

    bytes memory found2 = Suave.confidentialInputs();
    assertEq0(input, found2);
}

} ```
```

The value for the confidential inputs gets reset for each test.